Lessons Learned from a Collaborative Sensor Web Prototype

Troy Ames
NASA Goddard Space Flight Center

Lynne Case, Chris Krahe, Melissa Hess
Aquilent, Inc.

Abstract: This paper describes the Sensor Web Application Prototype (SWAP) system that was developed for the Earth Science Technology Office (ESTO). The SWAP is aimed at providing an initial engineering proof-of-concept prototype highlighting sensor collaboration, dynamic cause-effect relationship between sensors, dynamic reconfiguration, and remote monitoring of sensor webs.

I. INTRODUCTION

The Advanced Architectures and Automation Branch of NASA Goddard Space Flight Center developed the Sensor Web Application Prototype (SWAP) as an engineering proof-of-concept for technologies and architectures associated with sensor webs. A precise and generally accepted definition of a sensor web has not yet been formulated. However, it is generally agreed that sensor webs have certain intrinsic characteristics. In the context of this prototype, a sensor web consists of platforms that are interconnected by a communications medium so that they can exchange commands, status, and science data. The communications medium used was an Ethernet local area network (LAN). In our prototype, the instruments consisted of several tipping bucket rain gauges and a weather station equipped with an anemometer to measure wind speed and direction. Refer to [1] for some interesting applications and research in the area of sensor webs in the government and private sectors.

Some sensor webs may contain instruments and sensors, which simply perform measurements and report their data to a centralized computing platform. In this form of sensor web, at least one computing platform is collecting the data from all reporting sensors and synthesizing the data into one or more meaningful scientific products: a meteorological forecast model output for example. More sophisticated sensor webs consist of platforms that are equipped with instruments and sensors that communicate with each other in order to influence the behavior of one or more other platforms. For example, a rain gauge platform may cause a river gauge platform to begin to measure and report water levels at a higher frequency to determine if flash flood conditions may be imminent. This is a simple example of what has been called a "collaborative sensor web" and is depicted in Fig. 1. Collaborative sensor webs were the focus of our investigation for the SWAP project.

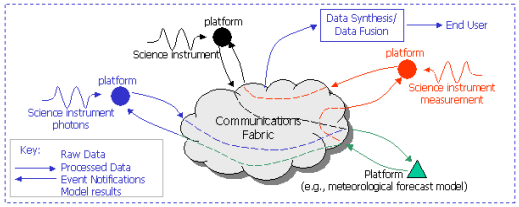

Fig. 1: Collaborative Sensor Web Overview

In a collaborative sensor web, a sensor must have some way of influencing the behavior of other sensors whether it be through data or explicit commands. This implies that the sensors have some "intelligence" to them in the form of data processing algorithms or heuristics that can be applied to the measurements they are taking. Many of the platforms that we investigated prior to developing the prototype were "dumb": they simply made their measurements and reported the "raw" data to a science processor. The concept of operation for these "dumb" platforms is thus very simplistic: the sensors are turned on, they collect data, and at some point a scientist retrieves the data where it can then be processed. In order to transform these "dumb" sensors into "smart" sensors and thus allow them to collaborate for our prototype system, we augmented them with microprocessors and developed embedded software applications to provide the required "intelligence" and thus with an ability to influence each others behavior.

The objectives of the SWAP project were to:

- Provide a proof of concept and feasibility of a sensor web.
- Identify near-term sensor web implementation issues and challenges.
- Identify technology gaps where additional research will be required to achieve sensor web goals.
- Assess the prototype's candidate software architecture.

II. SYSTEM OVERVIEW

The Instrument Remote Control (IRC) [2] software produced by the Advanced Architectures and Automation Branch was used as the primary software component within the sensor web. IRC is a software framework for monitoring and controlling instruments. Prior to the SWAP project, IRC had only been used to monitor and control astronomical instruments. One of the unique features of IRC is the ability of users and scientists to easily specialize the software for their instruments' needs. The software architecture that was assessed for SWAP involved using IRC to control and monitor the sensor web. Each of the sensor web platforms consisted of two components: an instrument that measured

either rainfall or wind velocity, and a microprocessor running an embedded version of the IRC software to process the rainfall or wind velocity data.

The Operating Missions as Nodes on the Internet (OMNI) project selected a variety of small processors for use with the sensors. The OMNI team configured these processors to run with various versions of Linux and provided them to the SWAP software development team for installation of IRC. The processors were connected to the sensors (i.e., rain gauges, anemometer) through serial computer port connections. Although the processors were equipped with wireless IP communications in anticipation that the prototype might be field deployed, an Ethernet LAN was used throughout the development and demonstration phases of the prototype.

A. Scenario

The Sensor Web Application Prototype Scenario Specification [3] provides a detailed description of the science scenario that was used to drive the SWAP system. A brief summary of this scenario is provided here. Fig. 2 depicts a layout of rain gauges and weather stations around Beaver Dam Road at the Beltsville Agricultural Research Center, located near GSFC. In the spring when this prototype would theoretically be deployed, the rain gauges and weather stations would be strategically positioned to take advantage of the typical weather patterns for that time of year and for that location. In the spring, the prevailing winds are expected to move from generally westerly directions (NW, W, SW) across the region. Thus when storms form and travel to the Beaver Dam Road region from those directions, the rain gauges positioned furthest west would be the first to detect the rain and thus be able to alert the other sensors in the system of imminent rain and wind conditions. Rain gauges positioned in the middle of the region would use the rain rate information as well as the wind speed and direction from a simulated radar system to predict the direction of the storm and the probability of its future appearance in their area. Finally, a "primary" rain gauge could synthesize the information from all other sensors to generate a prediction.

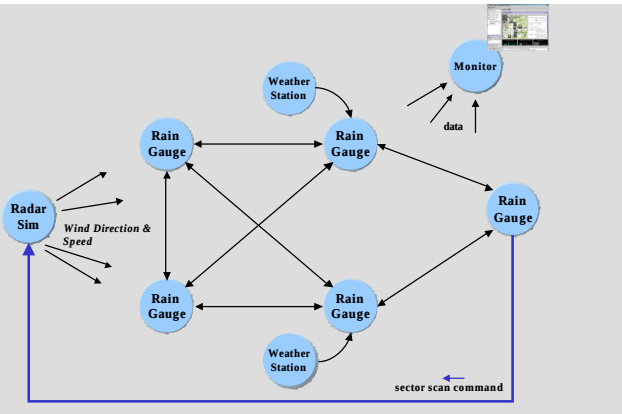

Fig. 2: Scenario Overview

This "primary" rain gauge was so called because it was uniquely located near the creek at Beaver Dam Road since that is the location where floods have frequently occurred during heavy rains.

The sensors communicate with each other and use information from each other to produce their own predictions and decisions. This is the collaborative nature of this sensor web. The main objective of this scenario was to send a "sector scan" command to a simulated radar system. This sector scan command would modify the mode of the radar from its nominal wide sweep mode of a broad region to a sector scan mode so that only the smaller region of intense weather conditions could be monitored and at a more frequent rate. The results of the sector scan mode of the radar system would be very useful to a meteorologist monitoring the storm system and especially the rainfall amounts and rate, and thus the likelihood of very rapidly rising water levels in the creek.

The Monitor circle in the upper right corner of Fig. 2 represents a graphical user interface (GUI) to monitor the sensor web. The GUI is not required to supporting the scenario, but it is a useful tool to display and monitor the real time events occurring at each rain gauge, the weather stations, and the radar simulator.

B. Architecture

Fig. 3 depicts the high-level architecture of the system. Each sensor (rain gauge or weather station) is connected to a small processor by a serial RS-232 cable. The processor has the IRC software running under the Linux operating system. The Radar simulator ran on a desktop PC. The PC in the lower right in the figure ran a weather station simulator since we only had one real weather station. Each of the systems were connected to the network using TCP/IP over Ethernet. In deployment they would use wireless IP.

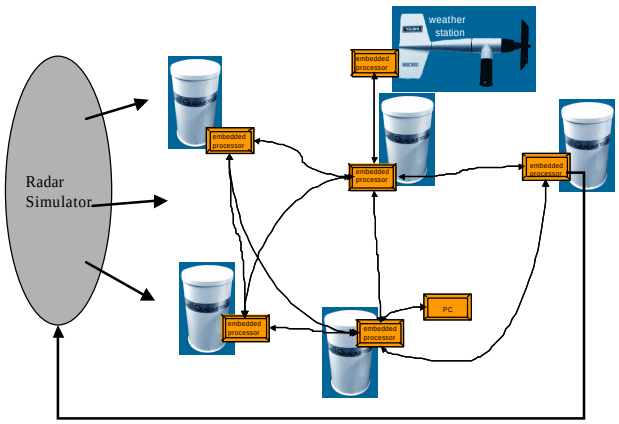

Fig. 3: High-level Architecture

The major challenge using IRC was to port it from a desktop environment to an embedded processor that could be fielded with the rain gauge and wind velocity instruments. A benefit of this architecture is that the measured data is processed and interpreted collocated with the instrument so that real-time decisions can be made based on the data the sensor is receiving. The software associated with each platform now provides it with the capability to locally process the measured data and to communicate these measurements and platform operational status with other platforms. The algorithms executed by the IRC software interpreted the data from the local sensor as well as other sensors in the sensor web in order to make decisions as described in the Scenario Overview.

III. HARDWARE

The computers, described in the following sections, were primarily chosen for their ability to run embedded applications and operate in an outdoor environment. This was an important consideration since it was intended that the sensor web might eventually be deployed to the Beaver Dam site. Physical dimensions, operating temperatures, and power consumption were therefore significant hardware selection criteria. By including these processors in our project, we learned some of their limitations and techniques for slimming down the IRC software that runs on them, as well as some practical lessons with Linux.

A. Tri-M MZ104

We used four MZ104-based computers in our prototype from Tri-M Systems (http://www.tri-m.com/). The MZ104 is a ZFx86, embedded pc-on-a-chip and was combined with other hardware, such as a disk-on-chip, on boards mounted in the shock-resistant “Can-tainer” pictured here.

We worked with Tri-M machines having two different configurations. One had 32MB of RAM and a 128MB disk-on-chip, while the other three Tri-M machines featured 64MB RAM and 64MB disk on chip. Physical dimensions were 91 and 274.6 cubic inches respectively (the smaller container was adequate).

Like all sensor computers in our prototype, the Tri-M communicated with its associated sensor via the serial port and with other sensor computers via Ethernet (using TCP/IP).

Unique to the MZ104 was the LinuxMZ operating system. Based on Slackware 7, LinuxMZ is a hardware-specific Linux variant shipped pre-installed on the device. After the OMNI team did some initial configuration with each Tri-M, the SWAP software team installed threading libraries, a Java virtual machine, a Linux-specific serial communications library, and other utilities.

B. IDAN PC-104

The IDAN PC-104 from Real Time Devices (http://www.rtdusa.com/), like the Tri-M, was designed to operate in embedded environments, though was more powerful. It featured a 233MHz Geode MMX processor running with 128MB RAM and a 2GB IDE disk. With this hardware, the IDAN ran Red Hat Linux 6.1 and the required Java virtual machine and serial communications library.

Consisting of independent modules hosting the processor, an Ethernet interface, disk, and power supply, the IDAN measures 100 cubic inches in size.

C. Matchbox

The Matchbox PC from Tiqit (http://www.tiqit.com/) promised the best packaging solution of all embedded computers used in our prototype. At only 5 cubic inches (and weighing 3.3 ounces), the Matchbox could easily be mounted inside one of our rain gauges.

In its current configuration, the Matchbox provided a 486SX processor, 1GB IBM Microdrive, and 16MB of RAM. It ran the Red Hat Linux 6.2 operating system, Java virtual machine, and serial communications library.

For the prototype, we used a breakout board to provide standard connectors to power, Ethernet, and serial ports. In a fielded situation, the board can be eliminated by connecting directly to the 68-pin female VHDCI (Very High Density Cabled Interconnect) connector.

D. Qualcomm Tipping Bucket Rain Gauge

The Tipping Bucket Rain Gauge, from Qualimetrics (http://www.qualimetrics.com/), was used to measure rainfall . The gauge features a simple see-saw like bucket at the bottom of a funnel which tips after 0.01 inches of rain. The tip closes a switch for 100 milliseconds, sending a pulse to the serial line connected to the sensor computer, where IRC counts the pulses to determine the rain rate. A custom serial cable was created to connect the rain gauge to the computer.

The rain gauge has an accuracy of 0.5% at 0.5"/hr. In windy conditions, the error is increased because the gauge cannot accurately collect and report all the rain that is falling (as some of the rain is blown over the top of the funnel). Algorithms within IRC were developed to account for this wind-induced error.

E. R. M. Young Weather Station

An R.M Young Weather Station (http://www.rmyoung.com/) provided wind data for our prototype. The weather station had other sensors attached to the unit butonly the wind speed and direction measurements were used for the sensor web prototype. Communicating via a supplied junction box and a custom serial cable we fashioned for this project, the weather station reported wind speed and direction in ASCII messages sent multiple times per second. IRC converted the units supplied by the weather station into meters per second and compass direction in degrees.

The sensor had an accuracy of ±0.3 rn/s (0.6 mph) for wind speed and ±3 degrees for wind direction.

IV. SOFTWARE

A. *Off-the-Shelf Libraries*

The Meteorological Applications (MetApps) is a Java library from the University Corporation for Atmospheric Research. The SWAP prototype used the MetApps library to calculate the distance and angle between two latititude and longitude points to predict the direction and location of the storm. It was also used to draw the wind barbs onto the weather map used in the Monitor GUI.

The IRC software was developed using the Java 2 Standard Edition from Sun Microsystems. Java provided the capability to easily port the IRC software to different operating systems. The machines used within the prototype were configured differently including varying amounts of memory and disk space. On machines with little disk space we needed to slim down the installed Java Runtime Environment (JRE). To reduce the needed disk space for the JRE we removed some Java libraries that were not needed such as the Java graphics toolkit.

IRC uses other COTS products such as the Apache Xerces parser, but those are not discussed in this paper since they were not specific to the SWAP project goals or development.

B. *Instrument Remote Control*

IRC is a framework for controlling and monitoring instruments that may be distributed across a network. The software architecture combines the platform independent processing capabilities of Java with the power of the Extensible Markup Language (XML), a human readable and machine understandable way to describe structured data. A key aspect of the architecture is that the software is driven by an instrument description, written using the Instrument Markup Language (IML), a dialect of XML. IML is used to describe the command sets and command formats of the instrument, communication mechanisms, format of the data coming from the instrument, and characteristics of the graphical user interface to control and monitor the instrument. Additionally, the IRC framework allows the users to define a data analysis pipeline, which converts data coming out of the instrument.

IRC provided a quick and easy way to get the sensors up in a network communicating with each other. In this context, a sensor is a single running instance of IRC in combination with either a physical sensor such as a rain gauge or a set of code to provide simulation routines.

To better understand what work was required to get a sensor up and running, we need to briefly describe how IRC uses IML. The main idea behind IRC and IML is to be able to describe an instrument and as a result have software that will allow you to communicate with that instrument. Fig. 4 is a high-level diagram that provides some insight into what is needed to describe a sensor.

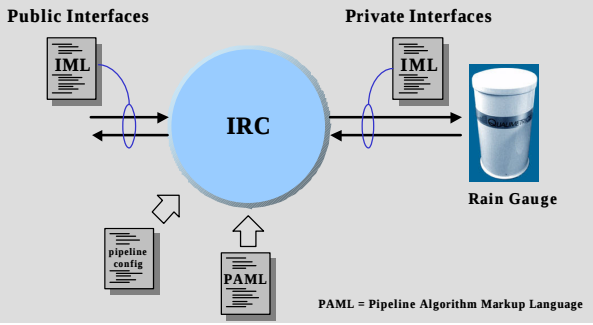

Fig. 4: Describing a Sensor

A single sensor can contain up to two IML descriptions as indicated in Fig. 4. The Private Interface describes how the sensor will communicate with other things in order to do its job. For example, a Rain Gauge Sensor which is depicted by the IRC circle above needs to communicate with the actual hardware rain gauge so that it can do things such as calculate the rate at which it is raining. The second IML description is called a Public Interface and it describes how something else can communicate with the Rain Gauge Sensor to receive rain rates.

In the configuration depicted by Fig. 4, the Rain Gauge Sensor will receive some type of data each time the actual rain gauge hardware tips, which signals 0.01 inches of rain has been collected. The Rain Gauge Sensor would then need to do some data analysis and execute an algorithm to determine the rain rate. IRC provides the Pipeline Algorithm Markup Language (PAML), an XML dialect, to plug in science algorithms that operate on the data. PAML provides the list of available custom algorithm types to IRC.

At runtime IRC provides a means to string two or more data analysis algorithms together into what is called a "pipeline". The beginning of this data analysis pipeline is data that is flowing into the sensor data ports. For the SWAP prototype each sensor had a predefined pipeline configuration executed at system start time. Fig. 5 shows the internal software configuration of a Rain Gauge Sensor.

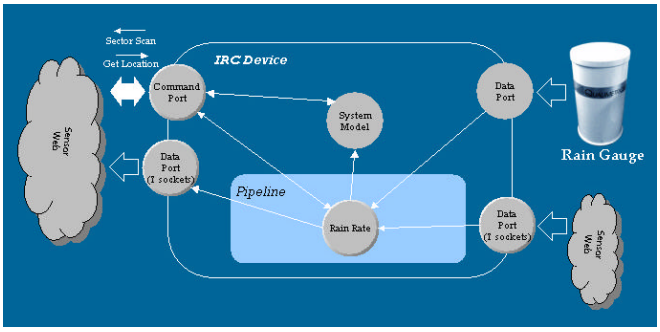

Fig. 5: Rain Gauge Sensor Internals

In our example the Rain Gauge Sensor is receiving the following types of data:

- Rain gauge hardware sends a pulse whenever the bucket tips.
- Another Rain Gauge Sensor is sending rain rates.
- Weather Station Sensor is sending ground wind speed and direction.
- Radar Sensor is sending atmospheric wind direction and speed.

In the case of the Rain Gauge Sensor there is only one algorithm. All the data coming in the data ports is sent to a Rain Rate algorithm. The Rain Rate algorithm can then produce predictions and rain rates. The Rain Rate algorithm output is then sent out a data port to any other sensor connected to the Rain Gauge Sensor.

1. Commands and IML

As described earlier, IML is used to describe instruments including the commands that an instrument accepts. IRC provides the framework to send a command to an instrument. In addition to IRC providing the framework to send commands to an instrument, IRC also has the capability to accept commands.

In the case of SWAP, neither the rain gauge hardware nor the weather station hardware accepted any commands; both simply transmitted data. In order to accept a custom command in IRC, first the command must be described in IML. Once the command is described, the logic to fulfill the command request is developed. The following describes the commands implemented for the SWAP prototype and how each command was used. We also describe the effort that is required to add custom command handling within IRC.

a) Get Sensor Location and ID –

Each sensor was located in some physical location. The location of each rain gauge and weather station was described in latitude and longitude. Additionally, each sensor in the web contained a unique identifier (ID). Typically, an ID would be something like RG1, representing rain gauge 1. Below is a simple and very basic sample of the IML required to add a command.

```
<Command name="Get Sensor Location and ID" />
```

IML allows for the specification of various information such as command arguments and synchronous versus asynchronous command execution. Later we will see what a command argument looks like.

The location command is most useful for monitoring the sensor web. The GUI that was used to monitor the state of the entire sensor web contained a map of the region surrounding the Beltsville Agricultural Research Center. In the scenario, the rain gauges and weather stations were positioned in this region. When the GUI started up, a command was issued to all of the sensors in the web to obtain the location of each sensor. Each sensor was able to provide a command response containing a location and unique ID. The location of each sensor was plotted onto this map.

b) Get Sensor System Time

Since each sensor in the web sent messages to all other sensors in the web, time synchronization was important when calculating data such as wind corrections. This command was a basic utility to quickly determine that the clocks on various machines were synchronized to within a second of each other.

c) Set Windspeed and Direction

The radar system used within the prototype was simulated. Below is a portion of the public IML file that described how to command the radar sensor.

```
<Command name="Set Windspeed and Direction" >
        <Field name="windSpeed" type="Integer" required="true"/>
        <Field name="windDirection" type="Compass"
                required="true"/>
</Command>
```

The command above was used to specify to the radar simulator the speed and direction of the storm it would generate. This command required two command arguments. The windSpeed command argument contains a name, type, and whether or not the argument is required for the command to be valid. The type specifies the data type of the command argument. For example, the windDirection value needs to be of type Compass. A Compass contains a value in degrees where 0 is north, 90 is east, and so on clockwise.

d) Sector Scan

The radar sensor also accepted a command to go into sector scan mode. This command was only sent to the radar sensor by the primary rain gauge sensor when its predicted rain rate or current rain rate exceeded some threshold rate. Recall from earlier that the primary rain gauge was called the primary because it is located near the creek at Beaver Dam Road that typically floods during heavy rains in that area.

e) Get Daily Measured Accumulation

Each rain gauge sensor accepted a command to retrieve the total rain during a single day. This command was created for testing and convenience purposes.

2. State Model

Each sensor in the web maintained some state information such as its location, unique ID, and system time. The IRC framework provides the capability to specify a custom state model that is constructed on startup. Since the state model was the keeper of information pertaining to several of the SWAP commands, it was a natural approach to use the state model to respond to several of the commands.

For SWAP, two state models were created. The more generic SWAP state model tracked the sensors location, id, and system time. The second state model was used for the

rain gauge sensors. The rain gauge state model extended the functionality provided by the generic SWAP state model to be able to track rain pulses. The rain gauge sensor was responsible for processing rain rates and predictions. In order to calculate rain rates, information from each rain gauge bucket tip was needed. Every time the rain gauge hardware tipped, the rain gauge sensor received a notification and stored the time at which the tip occurred. This information was stored in a state model so that an algorithm could make use of the data to determine individual rain events, rain event accumulations, and rain rates.

3. Weather Station Data Conversion

The SWAP prototype used one weather station. It reported several pieces of data including barometric pressure, temperature, wind speed, and wind direction. The data coming from the weather station was received over a serial connection and was in a raw form. The data needed to be converted to usable units. We added a custom algorithm that would take the raw data received from the weather station and convert it to meters/second rather than instrument cycles.

We chose to convert the weather station units to meters per second based on a survey of meteorological end user tools. The algorithm retrieved the wind speed and wind direction from the input key-value pairs. The wind speed raw value was in counts per 0.9994 Sec. There are 6 counts per revolution, and each revolution represents 29.4 cm of air movement. The wind direction raw value was already corrected by the compass reading, in tenths of a degree. The conversion was as follows:

*Wind direction (degrees)= (raw wind direction) * 0.1*

*Wind speed (meters per second) = (((((raw wind speed) / 6.0) * 29.4) / 0.9994) / 100.0)*

The algorithm would then output new key value pairs for wind speed and wind direction in meters per second and degrees respectfully.

4. Rain Rate Algorithm

The Rain Rate algorithm has several functions to perform. In general, its function is to determine rain event rain rates and predictions. The SWAP software team met with scientists from NASA GSFC's Mesoscale Atmospheric Processes Branch , Code 912, to determine how we should calculate rain rates. The scientists explained the notion of rain pulses and rain events. In summary, a rain pulse is considered to be a single bucket tip from the rain gauge. A rain pulse accounts for a specified amount of rain accumulation. In our case, each tip represented 0.01 inches of rain. A rain event represents a single rain "storm" and is comprised of a series of time stamped rain pulses. Typically, we only calculate a rain rate for a single rain event. Imagine if it rains early in the morning and then again in the late afternoon. Two separate rain events have occurred and therefore we would be interested in the intensity of each storm individually. A specified period of time is used to determine when one rain event ends and another begins. Once the specified amount of time occurs between rain pulses a new event is created. In a real operational environment, a half hour between pulses constitutes a new rain event.

The next factor to think about when determining rain rates is the ground wind. If the winds exceed a certain rate then the rain gauge is not collecting as much rain as is really falling. After consulting the NASA scientists we learned that adjusting the rain rates for the ground wind is very complicated and can potentially introduce errors in the data. With this in mind, it was decided that since this was a prototype about sensors interacting and making use of each others data, we would not worry about the absolute scientific correctness for ground wind correction. Rather, we came up with a simplistic way to adjust the rain rates to account for ground winds, which is discussed under Wind Corrections.

5. Predicted Rain Rates

The predicted rain rate algorithm receives data from all of the other rain gauge sensors in addition to the radar sensor. The algorithm has information about the movement of the storm based on the storm velocity (i.e. wind speed and direction) provided by the radar system simulator.

When the algorithm receives messages that it is raining at a remote rain gauge, it will look at the most recent data it has received from the radar sensor and determine if it is in the path of the storm. If it determines that it is in the path of the storm, it simply assumes that the rate at which it is raining at the remote rain gauge sensor is the rate at which it will eventually be raining at it. If it is not in the storm path then it has no predicted rain rate.

In addition to predicting a rain rate, it will also predict the time it will start to rain at its location. The message from the remote rain gauge sensor contains the remote sensor location. To determine when it will start raining at the local rain gauge, the algorithm will determine the distance between itself and the remote rain gauges. The algorithm makes use of the wind speed (meters per second) from the radar data to determine how long it will take for the storm to arrive. The algorithm applies a basic [(Rate * Time) = Distance] formula to predict the time when it will start raining.

6. Wind Corrections

Wind corrections are accounted for within the rain rate algorithm. When a rain gauge sensor receives data from a weather station sensor, it will determine if the wind data is applicable, based on the proximity between itself and the weather station. If the winds exceed a certain rate, then the rain gauge is not collecting as much rain as is really falling. The wind data is stored in the sensor's state model for use by the rain rate algorithm.

The SWAP prototype made some assumptions to greatly simplify the handling of ground wind data. The algorithm is not accurate from a scientific standpoint. The goal of the prototype was not to necessarily provide accurate meteorological algorithms, but rather to have different types of sensors sharing and making use of each others' data. A simple lookup table approach was used to store ranges of values with percentages of rate adjustments.

When the rain rate algorithm receives a new rain pulse, it will query the state model to determine if there is any wind data that could be applied to adjust the pulse amount. If wind data is available, then the average wind speed between the previous rain event pulse and the new pulse is calculated. The average wind speed and the parameter set are used to adjust the pulse amount. In the case where we have wind data but it is the first pulse in the event, we take the average of the wind data that precedes the new rain pulse by a few seconds.

7. Radar Simulator

Since the SWAP prototype was demonstrated in a controlled setting, we did not have access to a real radar system. We needed to have control of when it was raining and where the storm was moving to demonstrate the SWAP prototype.

The SWAP prototype constructed a radar simulator to play two key roles within the scenario. The radar simulator provided atmospheric wind direction and speed so the rain gauge sensors could predict movement of the storm. Additionally, the simulator accepted and responded to the sector scan command.

The SWAP team briefly investigated radar systems to determine the nature of the data that they produce. Based on the investigation, a radar simulator was built that was capable of parsing real vector wind profile files from the National Weather Service. We constructed a simulated data file so that we could control the movement of the storm.

IRC was used to represent the radar sensor and to also simulate the radar system. The radar data simulation was easy to create given the IRC pipeline and algorithm capabilities. We constructed a radar simulation algorithm by simply describing a new algorithm in PAML and then coding the logic. This algorithm stored state information, generated data when requested, and responded to the sector scan command when the sector scan command was received from the primary rain gauge.

8. Weather Station and Rain Gauge Simulators

In addition to simulating the radar system data, the SWAP prototype was also capable of simulating rain gauge bucket tips and raw weather station data. There are three reasons behind building these simulators. First, although the SWAP prototype was originally supposed to include two weather stations, we only received one. Therefore, to keep the scenario as originally planned, we simulated the second weather station. Second, the computers that were to be connected to the rain gauges and weather stations were received only a few weeks prior to the SWAP prototype demonstration. Thus, for testing purposes, the simulators provided a means to fully test the scenario without actually using true weather station and rain gauges. When the time came to actually connect all of the hardware contained in the prototype, the software was already tested.

3.10 Weather Map

The final piece of functionality that was added to complete the software for the SWAP prototype was a sensor web monitoring system. The monitoring system provides a graphical weather map to display status of the complete sensor web. Fig. 6 depicts the weather map monitoring system.

Fig. 6 is a complete IRC screenshot. On the left side of the image is the default IRC commanding tree. Each sensor is contained in the commanding tree, and the radar sensor commands are expanded. The default IRC GUI provided a means to configure the storm movement by issuing a "Set Windspeed and Direction" command to the radar sensor. The only custom portion of this interface is the weather map. The weather map algorithm received data from the rain gauge sensors, weather station sensors, and radar simulator in order to plot their locations and show the rain rates at each location.

V. LESSONS LEARNED

The SWAP prototype demonstrated that collaborative sensor webs using existing sensors can be developed. The sensors must be augmented with additional processing and software to use data from other sensors in the network. The following sections provide additional technical lessons learned relative to the actual implementation of the "smart sensors" using our candidate architecture.

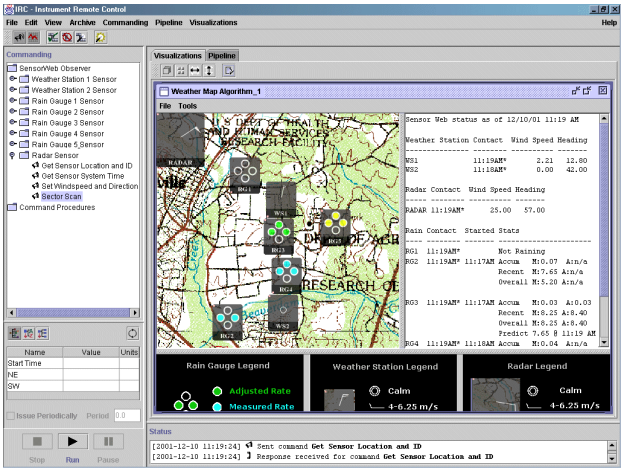

Fig. 6: Sensor Web Monitoring System

A. *Assessment of IRC-based Architecture.*

IRC provided the software architecture and framework that allowed for the SWAP prototype to be constructed and functional in such a short period of time. The complete prototype was functional within approximately three and half calendar months, using only 1.5 FTE (full-time equivalent) software engineers. Without IRC this would not have been possible. IRC is a viable component in a sensor web for the command and control of the individual sensors. Specifically, through the use of IML IRC provided the communications fabric that enabled the sensors to communicate and share data very quickly. Several key conclusions relative to using IRC within a sensor web are listed below.

- IRC provided a framework that made it easy to quickly communicate with the rain gauges and weather stations.
- IRC can be used to simulate new sensors / instruments. Adding simulators for the sensor web prototype proved to be very simple and extremely useful. The simulators provided a great way to test the software. As sensor webs are further explored it will be critical to be able to simulate instruments, because critical testing can be performed prior to the purchase of potentially expensive hardware.
- IRC can be made to run on small devices but additional work needs to be performed to tune the performance of the software in an embedded processor environment.

  For example, the sensors communicated with one another using TCP communications. TCP is a connection-oriented protocol and therefore slows communications. Another approach would be to explore UDP, which is a connectionless communication protocol. IRC provides the framework to add additional communication protocols.

  Additionally, XML parsing needs to be re-evaluated as a means to tune IRC performance. All of the XML and PAML files are parsed and validated each time IRC is started. It would be nice to remove the need to validate the documents. Validation during parsing consumes a large amount of memory during startup. If the files were developed using the IRC Configuration Editor, we know that they are syntactically correct, so we could skip the validation step during parsing.

  The number of threads needs to be reduced. The small processors could not handle the amount of memory and system resources that IRC was consuming. As an extensible architecture, there are many places for plugging in new modules. Many of these points of extension are accomplished through a publish and subscribe mechanism which uses a couple of threads per instantiation of subscribers. The use of threads needs to be evaluated to ensure that new threads of execution are used only when necessary.

B. *Implementation issues and challenges*

1. *Defining the algorithms is the greatest challenge.*

IRC provides the framework for constructing new algorithms and for passing information to other algorithms and instruments. The biggest challenge was to identify the appropriate science algorithms for the scenario that was simulated. To determine the characteristics of the algorithms to be used for the SWAP prototype, the software team consulted NASA scientists to determine the conditions that would establish when each individual storm begins and ends. Additionally, they consulted with scientists to determine how rain rates should be calculated and how ground winds affect rain rates. Currently, the process to adjust rain rates based on ground winds is being investigated, and does not have a definitive answer. Therefore, for the prototype we provided a simple approach that was not scientifically meaningful.

2. *Scalability*

The SWAP prototype consisted of only eight sensors, in addition to the monitoring GUI. The sensor web was small and thus provided a convenient testbed for determining how sensors could share data. As a sensor web grows in scale, the number of interactions will increase significantly, and it will certainly become more complex to study and evaluate the total number of processing and communications states that it can theoretically possess. A key problem moving forward will be planning for problems with many remote sensors communicating together toward a common goal. IRC does not currently address many issues that would arise in a large sensor web. For example, sensors will have to be added or removed without disrupting the entire sensor web. Network traffic loads will vary depending upon the number of available multiple modes of sensor-sensor interaction. IRC needs to be able to dynamically adjust to these mode changes instead of establishing a set of static conditions at startup time. During the demonstration, we eliminated some of the connections to enable IRC to run more efficiently and to minimize any possibility of running out of available resources (e.g., memory, disk storage) without further optimization of communication ports and protocols. In conclusion, although IRC was used successfully to prototype a small sensor web, it would not in its present form be able to support a large number of sensors that have limited resources (i.e., memory and disk).

3. *Porting IRC to Linux*

Porting IRC to Linux was easy. It simply required installation of two sets of libraries. First, we needed a Linux implementation of the Java CommAPI, which is used in IRC for serial port communications. The RXTX library (http://www.rxtx.org/), available under the LGPL license, provided this support. Second, we needed libraries to support native Linux threads, though only on the LinuxMZ machines (RedHat Linux already provided them). Since LinuxMZ is based on Slackware 7, we used libraries obtained from their site (http://www.slackware.org/).

4. *Sensor Web Weather Mapping*

Implementation of a weather sensor web would require a more sophisticated user interface. We recommend integrating an existing 3rd party application or library for this purpose. A candidate is MetApps (Meteorological Applications), some of whose components were used in this prototype.

C. *Technology Gaps*

The variety of sensor computers we employed presented us with various challenges, successes, and failures.

- The IDAN machine performed flawlessly throughout the project, keeping up with the volume of data, algorithms, and communications with other sensors.
- The Tri-M machines gave mixed results. During installation we trimmed the JVM and IRC to fit on the 64MB disk. While testing we experienced slow startup times (~3 minutes), intermittent errors in the Linux threading library, and occasional system failures due to static electricity.
- With only 16MB of RAM, the Matchbox PC could not keep up with the demands of standard Java and IRC. It rarely reported its data to the sensor web, even after minimizing the number of connections it needed to make to other computers. Despite this performance, the Matchbox was reliable.

As a result, we identified the optimum configuration for current SWAP implementation as 64 MB RAM and 128 MB disk. When taking into account its size, reliability, and ability to run standard Red Hat Linux, future iterations of the Matchbox would be ideally-suited. As of this writing, however, a 32MB RAM Matchbox PC is the next planned release.

There are other approaches to reduce SWAP's hardware requirements. Research areas include the following:

- Test with Java Standard Edition 1.4 – it reduces the need for a thread per TCP/IP connection; this may reduce runtime resource needs and allow for full communication between all sensors
- Identify tuning opportunities within IRC – startup times & memory usage for IRC might be reduced, for example, by optimizing or replacing the XML parsing with another approach
- Experiment with Java 2 Micro Edition -- the CDC (Connected Device Configuration) with the Foundation Profile is the best place to start our research. Together, they are the closest to the J2SE feature-wise, and are also only one of two combinations available for Linux at the moment (at least from Sun). The CDC is intended for 32-bit microprocessor/controller with more than 2.0MB of total memory.

ACKNOWLEDGMENT

We thank Dr. Marshall Shepherd for working with us to develop the initial science scenario and vision for the prototype. Mr. Brad Fisher, Dr. Eyal Amitai, and Dr. Ali Tokay consulted with us about algorithms for tipping bucket rain gauges and wind correction. They also provided us with loaner rain gauges to test and demonstrate our system. James Rash (NASA), Ron Parise (CSC), Keith Hogie (CSC), and Ed Criscuolo (CSC) worked on the hardware and provided us with our loaner weather station as well as many cables. Xuewu Cai, Anass Manjra, Tony Ritrivi, and Mike Mersky of Aquilent helped construct and test the SWAP prototype and supported the demonstration. Last but not least, Julie Breed, the Code 588 Branch Head at GSFC, and Steve Talabac of Aquilent provided us with guidance and editing of content for our documentation, demonstration, and web site.

REFERENCES

[1] Steve Talabac, Lynne Case, Mike Mersky (designer) "GSFC ISC Sensor Web Research", web page pointing to Sensor Web research around the world as well as the subject material of this paper. http://pioneer.gsfc.nasa.gov/public/sensorweb/

[2] Troy Ames, "Instrument Remote Control Project" web site at: http://pioneer.gsfc.nasa.gov/public/irc

[3] Lynne Case, "Sensor Web Application Prototype Scenario Specification", obtain at: http://pioneer.gsfc.nasa.gov/public/sensorweb/SWAP Prototype.doc